価値関数によるソフトリアルタイムシステムのスケジューラ自動合成手法(計算機科学の理論とその応用)

AUTHOR(S):

加納, 卓; 山根, 智

CITATION:

加納, 卓 ...[et al]. 価値関数によるソフトリアルタイムシステムのスケジューラ自動合成手法(計算機科学の理論とその応用). 数理解析研究所講究録 2007, 1554: 186-193

ISSUE DATE:

2007-05

URL:

http://hdl.handle.net/2433/80954

数理解析研究所講究録
第1554巻 2007年 186-193

価値関数によるソフトリアルタイムシステムのスケジューラ自動合成手法

金沢大学・自然科学研究科　加納卓 (Suguru Kano)　山根智 (Satoshi Yamane)
Graduate School of Natural Science & Technology,
Kanazawa University

1 序論

リアルタイムシステムはリアルタイムオペレーティングシステムとともに動作して，信頼性保証が難しいシステムであり，仕様記述と検証は重要である [1][2]．本研究では，Controller Synthesis パラダイムを用いて，ソフトリアルタイムシステムのスケジューラ設計手法を提案する．

２００２年，Sifakis らによって Controller Synthesis パラダイム [3] に基づいた，ハードリアルタイムシステムのスケジューラ設計手法が提案されている [4]．なお，Controller Synthesis パラダイムとは，スケジュールすべきシステムの時間モデルと，スケジュール要求を示す制約式からスケジューラを構築するためのフレームワークである．

本論文では，価値関数により Sifakis らの手法を改良して，ソフトリアルタイムシステムのスケジューラ設計手法を提案する．この手法は，Sifakis らの提案した手法と，以下の点において異なる．

1. Sifakis らの用いた”時間システム”に価値関数 [5] を付加した”価値関数つき時間システム”を用いて，システムのモデル化を行う．
2. ２つ以上の制御可能アクションが実行を待っている際，将来的にシステム全体の価値が最大になるよう，アクションを制御する．

なお，今回は初期状態から始まり，初期状態に戻ってくるシステムを扱う．これは，実行列を求める際，全てのプロセスが初期状態に戻ってくるまでの実行列を求めるためである．それに伴い，価値関数付き時間システムのシンタックスには初期状態を示す項を含めた．

関連研究として，以下のものが挙げられる．1987年，P.Ramadge らによって，離散事象システムを対象として，環境とシステムとの動作をゲーム理論としてモデル化して，その勝つ戦略を求めるアルゴリズムが提案された（Controller Synthesis パラダイム [3]）．そして，時間オートマトン上でその理論を用いたものとしては，Wong-Toi らによる，言語ベースのアプローチによるもの [7] と，J.Sifakis らによる，状態ベースのモデルによるもの [8] が挙げられる．本論文で提案する手法の基となったスケジューラ設計手法 [4] は，J.Sifakis らの，ハードリアルタイムシステムのスケジューラ合成手法である．本論文で提案する手法は，それをソフトリアルタイムシステムに対応できるように拡張したものである．

本論文では，まず２節において，価値関数，価値関数付き時間システム，制約式について述べる．次に，３節において，スケジューラを構成するための方法論，及び，価値を最大化するスケジューリングについて述べる．最後に，４節において，結論を述べる．

2 ソフトリアルタイムシステムのモデル化

本節では，Sifakis らによるハードリアルタイムシステム向けのモデル [4] を，ソフトリアルタイムシステム向けに拡張したものについて述べる．まず，ソフトリアルタイムシステムの性質を表現する”価値関数”について述べ，次に，それを導入したモデル”価値関数付き時間システム”について，最後に，そのアクションを制限するための”制約式”について述べる．

2.1 価値関数

ソフトリアルタイムシステムにおいて，従来のハードリアルタイムシステムにおけるデッドラインのようなものはなく，デッドラインミスによるシステムの停止は無い．今回は，システムにプロ

セスが完了するまでにかかった時間によってそのプロセスの価値が変化する性質を与え，それによりシステム全体の価値を評価することで，ソフトリアルタイムシステムの性質を表現する．その特性を実現するため，”価値関数”を用いる [5]．
価値関数を以下のように定義する．

Definition 1 (価値関数) t をプロセスの到着からの経過時間とするとき，

$V(t) = value$　　($value$ はプロセスの価値)

□

2.2 価値関数付き時間システム

時間システムとは，タイマ変数が実数範囲である時間オートマトン [9] やハイブリッドオートマトン [10] のように，オートマトンをタイマ変数で拡張したものである．本研究では，それに価値関数を付加した，”価値関数付き時間システム”を用いる．以下”時間システム”と書くときは，全て”価値関数付き時間システム”を指すものとする．

時間システムでは，プロセスをモデル化するために，制御可能アクションと制御不可アクションの２つの種類のアクションを持つ時間システムを用いる．

制御可能アクション　スケジューラが実行を指示できるアクション（例，リソース割り付け等）

制御不能アクション　スケジューラが実行を指示できない，外部環境からのアクション（例，プロセス到着など）

Definition 2 価値関数付き時間システム $TS = (S, A, T, s_0, X, V, \mathbf{b}, h)$ は，以下の *1*〜*6* で構成される

1. ラベル遷移システムの３つ組 *(S,A,T)*，ここで
   S: 状態の有限集合
   A: アクションの有限語彙
   T: 遷移関係 $(T \subseteq S \times A \times S)$
2. 初期状態 s_0
3. タイマ変数の有限集合 X
4. 価値関数の有限集合 V
5. 関数 $\mathbf{b}$　$\mathbf{b}$ は各状態にタイマ変数が進むかどうかを表す *boolean* 配列を割り当てる関数
6. ラベリング関数 h　h は遷移関係にガードやリセットする変数の集合，ループ評価式，価値関数，価値関数の引数となるタイマ変数を割り当てる関数 $h(s,a,s') = (s,a,g,\tau,r,l,f,s')$
   ここで，
   g: ガード
   τ: 実行タイミングが来たらすぐ実行するかを示すものであり，以下のとおりである．

$$\tau = \begin{cases} \delta & (delayable : \text{実行タイミング中ならいつでも実行可}) \\ \epsilon & (eager : \text{実行タイミングが来たらすぐ実行}) \end{cases}$$

   r: リセットするタイマ変数の集合
   l: ループ評価式 $loop \leq max$，(max は最大ループ回数)
   v: (v,t)，$v \in V$，t はタイマ

□

意味　時間システムから，頂点と枝によって構成される遷移グラフ $(\mathcal{V}, \mathcal{E})$ を構成することが出来る．

$\mathcal{V}$: 時間システムの状態を表す．状態 S とタイマ変数の値の列で表現される $(\mathcal{V} = S \times \mathbf{R}^{|x|})$

$\mathcal{E}$: 時間システムの状態遷移を表す．
$(\mathcal{E} = \mathcal{V} \times A \cup \{\mathbf{R} \setminus \{0\}\} \times \mathcal{V})$

$\mathcal{E}$ は，$\mathcal{E}^c$（制御可能），$\mathcal{E}^u$（制御不能），$\mathcal{E}^t$（時間遷移）の３つに分類される．$\mathcal{E}^c, \mathcal{E}^u$ は，オートマトン上の状態が変化し，タイマ変数が必要に応じてリセットされる遷移である．そして，$\mathcal{E}^t$ は，オートマトン上の状態が変化せず，タイマ変数のみが進む遷移であるが，現在の状態から伸びる，全てのアクションについて，そのガードが $\tau = \delta$ であるもの全てが実行不能にならず，そのガードが $\tau = \epsilon$ であるもの全てが実行不能であるという条件を満たしたもののみが，それに含まれる．□

2.3 価値関数付き時間システム例

図１に，基本的なプロセスの例を示す．
このプロセスは，３つの状態 s, w, e と３つのアクション a, b, e, l を持つ．アクション b のみ，制御可能アクションである．なお，制御不能アクションには，上付き文字 u を付加する．タイマ変数 x は実行時間を測り，タイマ変数 t はプロセス到着からの経過時間を測る．２つのタイマ変数は，全ての状態において経過する．アクション l にはループ評価式によって，最大ループ回数が３回であることが示されている．また，アクション e には，価値関数 v とタイマ変数 t の組が割り当てられている．すなわち，このアクションが実行されるタイミングで，タイマ変数 t の値を引数に，価値関数 v によって価値が判定される．□

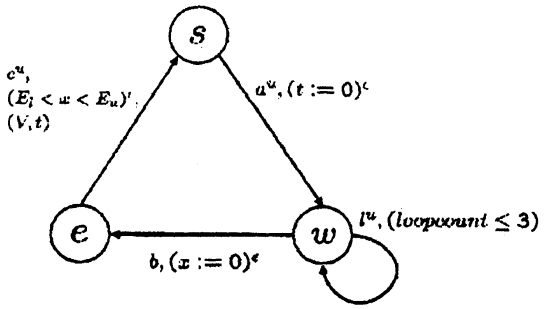

Fig. 1: プロセス例

Definition 3 (並列合成) 本論文では，*Sifakis* らによるものに，価値関数の要素を加えた．価値関数付き時間システムの並列合成を定義する．$TS_i = (S_i, A_i, T_i, s_0^i, X_i, V_i, \mathbf{b}_i, h_i)$ $(i = 1, \cdots, n)$ を時間システムの集合とし，Σ を同期アクションの集合とする．

TS_i を並列合成して作られる時間システム $TS = (S, A, T, s_0, X, V, \mathbf{b}, h)$ は，次のように定義される．

$$S = S_1 \times S_2 \times \cdots \times S_n$$
$$A = A_1 \cup A_2 \cup \cdots \cup A_n \cup \Sigma$$
$$s_0 = (s_0^1, s_0^2, \cdots, s_0^n)$$
$$X = X_1 \cup X_2 \cup \cdots \cup X_n$$
$$V = V_1 \cup V_2 \cup \cdots \cup V_n$$

すべての $s = (s_1 \cdots s_n) \in S, x \in X$ について

$$\mathbf{b}_s[x] = \begin{cases} 0 & (\mathbf{b}_{i,s_i}[x] = 0 \text{ となる} \\ & \quad \mathbf{b}_{i,s_i}[x] \text{ が存在するとき}) \\ 1 & (\mathbf{b}_{i,s_i}[x] = 0 \text{ となる} \\ & \quad \mathbf{b}_{i,s_i}[x] \text{ が存在しないとき}) \end{cases}$$

なお，$\mathbf{b}_s[x] = 0$ とは，状態 s において，タイマ x は増加しないことを示す．

すべての $s = (s_1 \cdots s_n) \in S$, $s = (s_1' \cdots s_n') \in S$ について，以下である．

（インターリービング遷移） 以下の条件を満たすとき，アクション $a \in A_i$ はインターリービングアクションとして合成後の TS の遷移関係 T に (s, a, s') として含まれる．

1. s について，s_i のみ s_i' に変化し，それ以外の $s_j (j \neq i)$ については変化しない．
2. すべての Σ の要素について，a が含まれるものが存在するなら，その同期アクション集合の中に現在の状態から行われるアクションではないものが含まれる．

このとき，ラベリング関数の g, τ, r, l, v はそのアクションのものを抽出する．

（同期遷移） 以下の条件を満たすとき，アクション $\mathbf{a} \in \Sigma$ は同期アクションとして合成後の TS の遷移関係 T に $(s, \mathbf{a}, s')$ として含まれる．

1. $\mathbf{a}$ に含まれる a_i のすべてに対して，対応する TS_i に遷移関係 (s_i, a_i, s_i') が存在し，含まれるアクションがない $TS_j (j \neq i)$ については対応する s_j は変化しない．
2. $\mathbf{a}$ を完全に含むすべての $\mathbf{a}'$ について，現在の状態からの遷移ではないアクションが $\mathbf{a}'$ に含まれる．

このとき，ラベリング関数の g, τ, r は，$\mathbf{a}$ に含まれるすべてのアクションに対して，以下となる．

g : それらのアクションの g の選言，

τ : それらのアクションの τ の中にひとつでも ϵ が含まれれば ϵ，

r, l, v : それらのアクションの r, l, v の和

2.4 制約式 (Constraint)

制約式は，それで時間システムを制限することによってリアルタイムアプリケーション上のスケジューラの効果をモデリングするために用いられるものである．制約式を満足させるため，スケジューラは制約式を満たす状態へ続くときのみ，その制御可能アクション実行を指示する．

Definition 4 (制約式の一般形 [4]) 本論文では，*Sifakis* らの用いたもの *[4]* と同じ定義を用いるものとする．X をタイマ変数集合，S を状態集合，T を遷移関係の集合とする．制約式は以下の式で表される．

$$K = \bigvee_{s \in S} s \wedge K^s \wedge \bigwedge_{(s,a,s') \in T} [a](K^{sas'})$$

K^s は状態に付くタイミング制約，$K^{sas'}$ は遷移関係に付くタイミング制約，s はブール変数で，状態 s に存在することを示す． □

時間システムの制約式による制限は，すべての制御可能遷移 (s, a, g, τ, r, s') のガード g を，以下の式のようにして g' で置き換えることで行われる．

$$g'(\mathbf{x}) = g(\mathbf{x}) \wedge K^{sas'}(\mathbf{x}) \wedge K^{s'}(\mathbf{x}[r])$$

なお，時間システムTSを制約式Kで制限したものをTS/Kと表す．

3 スケジューラの構成方法論

本節では，2節で述べたモデルにより表現されたシステムについて，実際にスケジューラを構築する手法について述べる．Sifakisらによる従来の方法[4]では，スケジューラの設計は，リソースの相互排他などの単純なものを除いて，設計者が独自に制約式を考える必要があったが，本論文で提案するスケジューラ設計法では，価値関数から自動的にスケジューラの設計，すなわち制約式の導出を行うことができる．

本節の以下において，まず，スケジューラを構築するための準備として，"不変式"について述べ，次に，スケジューラを構築する方法論について述べる．続いて，スケジューラを設計するために，モデルに記述するべき情報について述べ，スケジューラが正しくシステムを制御できているかを判定するための"デッドロック要求"について，そして，スケジューラを設計するための"スケジュール要求"について述べる．最後に，上述した，価値関数から自動的にスケジューラを設計するための手法，アルゴリズムについて述べる．

3.1 準備

不変式とは，時間システムを制限する処理において重要なものであり，次の2種類が存在する．特に**コントロール不変式**かどうかを判定することは，制御可能アクションのみを制限することで要求を実現するために非常に重要な判定である．なお，これらの定義は，Sifakisらの用いた定義[4]をそのまま用いている．

Definition 5 (完全不変式 [4]) TSを時間システム，Kを制約式とする．制約式Kが時間システムTSの完全不変式であることは，$T \models inv(K)$で表される．TSの全ての遷移によってKが守られているならば，KはTSの完全不変式である．すなわち，以下の式で表される．

$$\forall(s,\mathbf{x}) \cdot (K^s(\mathbf{x}) \wedge \exists((s,\mathbf{x}),\gamma,(s',\mathbf{x}')) \in \mathcal{E}) \Longrightarrow K^{s'}(\mathbf{x}') \wedge (\gamma \in A \Longrightarrow K^{sas'}(\mathbf{x}))$$ □

Definition 6 (コントロール不変式 [4]) TSを時間システム，Kを制約式とする．$TS/K \models inv(K)$ならば，Kはコントロール不変式である．すなわち，制約式Kが制御可能アクション以外による全ての遷移によって守られるならば，Kはコントロール不変式である． □

3.2 制御可能アクションの制約式の構成

制御可能アクションの制約式は，いわばスケジューラの外部仕様のようなもので，スケジューラがシステムにその制約式に示される制限を守らせることによって，システムの動作にルールを強いることのできるものである．なお，以下に示される手順は，Sifakisらの用いた手順[4]と同じであるが，スケジュール要求を導出する手法は，本研究のオリジナルである．それについては後述する．

方法論 制御可能アクションの制約式は，以下の手順で構成される．

1. システムの各プロセスを時間システムTSでモデル化する．
2. 作成したTSを並列合成し，システムのモデルとする．
3. デッドロック回避要求K_{ndead}を求める．
4. システムに対する様々なスケジュール要求$K^i_{pol}(i=1,\cdots,n)$を制約式で表す．
5. （4）で作成したすべてのK^i_{pol}について，以下の操作を実行．
   (a) K^i_{pol}がTS'のコントロール不変式(または完全不変式)であるかチェックする．
   (b) (a)の結果が真ならばTS'をK^i_{pol}で制限し($TS''=TS'/K^i_{pol}$とし，TS''をTS'とおく)，(a)に戻る．
   (c) (a)の結果が偽ならば$K^i_{pol} \wedge K^{i+1}_{pol}$が$TS'$のコントロール不変式であるかチェックする．
   (d) (c)の結果が真ならばTS'を$K^i_{pol} \wedge K^{i+1}_{pol}$で制限し($TS''=TS'/K^i_{pol}$とし，$TS''$を$TS'$とおく)，(a)に戻る．
   (e) (c)の結果が偽ならば次のスケジュール要求について(c)以降の操作を繰り返す．
6. K_{ndead}が最終的に求まった時間システムの完全不変式であるかチェックする．
   (a) 結果が真ならば求まった制御可能アクションのガードがシステムを正しく制御する制約式となる．
   (b) 結果が偽ならばシステムがデッドロックを起こす可能性があるので失敗．

3.2.1 システムのモデリング (1)

適切なスケジュールを行うためには，システムの様々な情報が必要となる．そのため，システムのモデル化の段階で，次の情報を記述する．

状態 プロセス p_i の状態に対して，次の２つの述語を定義する．

1. $U_{i,r}(s)$: プロセス P_i は状態 s 上でリソース r を使用する．
2. $W_{i,r}(s)$: プロセス p_i は状態 s 上でリソース r を待つ．

アクション プロセス p_i のアクションを，以下のように分類する．

1. *Arrival*: 休止状態 (sleeping) から待機状態 (waiting) へ行くアクション．
2. *Begin*: あるリソース r について，$W_{i,r}(s) = true$ な状態 s から $U_{i,r}(s') = true$ な状態 s' へ行くアクション．$BGN_{i,r}$ はこのアクションの集合．
3. *End*: $U_{i,r}(s) = true$ な状態 s から $U_{i,r}(s') = false$ な状態 s' へ行くアクション．$END_{i,r}$ はこのアクションの集合．

タイミング制約 プロセス p_i に次の変数を追加．

1. timer t_i: プロセス到着からの経過時間を計る．到着アクション (*Arrival*) によってリセットされ，増分は常に１．
2. timer x_i: 実行時間を計る．リソース r を要求する開始アクション (*Begin*) によってリセットされ，$U_{i,r}(s) = true$ な状態 s 上で増分は常に１．

これらのタイマ変数は次の２つのタイミング制約式を表現するために用いられる．

Inter-arrival constraints. プロセスの到着周期の上下限の制限．

(式例) $T_i^l \leq t_i \leq T_i^u$

Execution time constraints. 実行時間の上下限の制限．

(式例) $E_{i,r}^l \leq x_{i,r} \leq W_{i,r}^u$

3.2.2 デッドロック回避要求 (3)

デッドロック回避要求とは，各プロセスが必ず守らなければならないタイミング制約を示したものである．なお，この制約式は制御不能アクションのタイミング制約を含めて導出されるので，この制約式でシステムを制限することはできない．これは，あくまで最終的にシステムが正しく制御されているかをチェックするためのものである．

Definition 7 (◇ の定義 [4]) C を制約式，$s \in S$ をプロセスの状態とする．全ての $\mathbf{x} \in \mathbb{N}$ について，

$$(\Diamond_s C)(\mathbf{x}) = \exists t \geq 0 \cdot C(\mathbf{x} + t\mathbf{b}_s)$$

□

$(\Diamond_s C)(\mathbf{x})$ は，state s において time が無期限に進むことができるなら，いつかは $C(\mathbf{x})$ が満たされる，という意味である．

Definition 8 (デッドロック回避要求 [4]) i はインデックスとする．与えられたプロセス P_i の，結合されたデッドロック回避要求 K_{ndead}^i は以下の式で与えられる．

$$K_{ndead}^i = \bigvee_{s=(s_1,\cdots,s_n)\in S} s_i \wedge \Big(\bigvee_{(s_i,a,s_i')\in T_i} \Diamond_s g_a \Big)$$

ここで，g_a は，遷移関係 (s_i, a, s_i') のガードである．

□

K_{ndead}^i が不変的に守られるということは，何らかのアクションを実行することがいつでも可能である，すなわちデッドロックフリーであるということを示す．なお，対象となる時間システムがこの要求を満たしているかのチェックは，モデル検査ツールとして UPPALL[11] を用いて行う．

3.2.3 スケジュール要求 (4)

スケジュール要求とは，システムに対する要求を満たすための，制御可能アクションの制約式である．スケジュール要求を作成する事は，いわば，スケジューラを設計する事に相当する．

スケジュール要求は，リソース許可制御 (Admission Control) と，衝突回避 (Conflict Resolution) の２種類に区別される．

リソース許可制御 リソース許可制御とは，いくつかあるプロセスのうち，どのプロセスがリソースを使用する資格を持つのかを決定するための制約式である．

Definition 9 (リソース許可制御 [4]) リソース許可制御 $K_{adm} = \wedge_{r\in R} K_{adm}^r$ は，以下の式によっ

て示される遷移制約式によって特定される.

$$K^{T}_{adm} = \bigvee_{s \in S} s \wedge \bigwedge_{(s,a,s') \in T, \exists i, a \in AL_{i,r}} [a](K^{sas'})$$

□

タイミング制約 $K^{sas'}$ は，リソース r をプロセス P_i に割り当てる遷移 (s,a,s') にラベル付けされたすべてのアクション a を制限する．この式において，$AL_{i,r}$ とは，プロセス P_i にリソース r を割り当てる時間システムのアクションである．

衝突回避　衝突回避とは，２つ以上のプロセスのリソースの衝突を，優先度を適用させることによって解決するものである．衝突回避のために，システムの動作を制限する優先度を導入する．

優先順位は，半順序 $\prec \subseteq A^c \times A$ である．(a_1, a_2) が $\prec$ に含まれることを，$a_1 \prec a_2$ と書く．

Definition 10 (優先規則 [4]) 優先規則とは，優先度を決める規則である．優先規則は組 $pr = (C^j, \prec^j)_{j \in J}$ の集合である．このとき，$\prec^j$ は優先順位 C^j は優先順位が適用される時刻を特定する状態制約式である．

優先規則を適用させる制約式 K_{pr} は次の遷移制約式である．

$$K_{pr} = \bigvee_{s \in S} s \wedge \bigwedge_{(C,\prec) \in pr} \bigwedge_{a_j \in A} [a_j](\neg C(s) \vee \bigwedge_{i \in I, a_j \prec a_i} \neg g_i)$$

□

優先規則を適用した時間システムは TS/K_{pr} と書く．

3.3 価値を最大化するスケジュール要求

今回，ソフトリアルタイムシステムを対象とするに当たって，導入した”価値関数”を用いたスケジューリングを行う事で，システムの価値を最大化するためのスケジュール要求を導出する手法を提案する．

価値関数によるリソース許可制御　プロセスに価値の下限値を与える価値要求を定義する．

Definition 11 (価値要求) V_{P_i} をプロセス P_i の価値関数，t_{P_i} を価値関数 V_{P_i} の引数となるタイマ変数とする．プロセス P_i に評価値 $value$ を要求する価値要求は次の式のようになる．

$$vr^{P_i} = V_{P_i}(t_{P_i}) \geq value \ (value \text{は任意の価値})$$

この価値要求を満たすための制御可能アクションの制約式 $K_{vr^{P_i}}$ は，タイマ変数 t_{P_i} に関するタイミング制約として，価値関数 V_{P_i} から求められる．

□

システムのモデルを制約式 $K_{vr^{P_i}}$ で制限することで，システムに価値要求として与えた下限値を守らせることが出来る．

価値関数による衝突回避　プロセスの価値を最大化する時，一つのプロセスのみに注目するなら，先に定義した価値要求のように単純なタイミング制約で実現できる．しかし，リソースの競合が発生しており，どのプロセスにリソースを許可するかを決定するような場合は，単純な方法で導出することはできない．そこで，そのような状態から始まる全ての実行列を求め，列ごとに価値を算出し，その値の最も高くなるプロセスにリソースを与えることで，価値を最大化するための，リソースの衝突回避制約を導出する．

これは，実行列と予測価値の導出と，制約式の導出の２ステップで行われる．

ステップ１：実行列と価値式の導出　実行列として，起こり得るあらゆる動作を考えて，その列を全て列挙する．そして，列挙された実行列ごとに，価値式を各プロセスの到着からの経過時間を計るタイマ変数（プロセスタイマ）の探索開始時の値に関する式として求める．ここで，新たにタイマ変数をプロセスの数だけ追加し，それを用いて各プロセスの探索開始時からの経過時間を計る [12][13]．価値式から価値を計算する際，価値関数と，探索開始時のプロセスタイマの値，探索開始時からの経過時間から価値を求めるが，制御不能アクションの実行タイミングは範囲を持って指定されているので，各プロセスの探索開始時からの経過時間は範囲を持つことになる．そこで，価値は計算上，範囲の中で最も低い値をとるとする．これは，制御不能アクションは価値を下げる方向に働くと仮定することで，システムの最低価値を最大化するための条件を求めるためである．[14]

アルゴリズムは，以下の様になる．

Definition 12 (アルゴリズム)

```
VALUE_EXPS output_values; // 価値式の出力
PATHS      output_paths;  // 実行列の出力

// システムの時間システム上を深さ優先探索
// することによって，初期状態に戻るまでに
```

```
// 実行可能な全ての実行列を列挙する関数.
void PATH_SEEK(
  STATE st // 探索元の状態
  ) {
  TIMER     t[number of process];
  // 探索開始時のプロセスタイマ変数の値
  TIMER     p[number of process];
  // プロセス毎の探索開始時からの経過時間
  PATH      path;  // 探索中の実行列
  VALUE_EXP value; // 探索中の価値式

  // 現在の状態から伸びている全ての制御可
  // 能アクションについて，それを実行した
  // 後の状態と，実行したアクション，タイ
  // マ変数，経過時間，実行列，価値式を引
  // 数に，関数 PATH_SEEK_LOOP を呼び出す.
  forall CONTROLLABLE ACTION ca from st {
    PATH_SEEK_LOOP(st(ca),ca,t,p,path
                   ,value);
  }
}

// 再帰呼び出しによって，システムの時間シ
// ステム上を実際に探索する関数.
void PATH_SEEK_LOOP(
  STATE     st,    // 現在の状態
  ACTION    a,     // 実行したアクション
  TIMER     t,     // プロセスタイマ
  TIMER     p,     // プロセス毎の経過時間
  PATH      path,  // 探索中の実行列
  VALUE_EXP value  // 探索中の価値式
  ) {

  // 初期状態に戻っていないかチェックする
  // 戻っているなら，実行列，価値式を出力
  // 先に追加し，再帰呼び出しを終了する.
  if(st.IsInitialState) {
    output_values.add(value);
    output_paths.add(path);
    return;
  }

  // 実行したアクションが価値を判定するア
  // クションなら，そのプロセスの価値関数
  // と，探索開始時のプロセスタイマの値，
  // 経過時間を示すタイマ変数から，価値式
  // を導出し，探索中の価値式を更新する.
  if(a.IsValidationAct) {
    value.add_exp(a.value, t[st.process],
          p[st.process]);
  }

  // 現在の状態から伸びる全てのアクション
  // について，以下の引数を渡して再帰呼び
  // 出しする.
  forall action a from st {
    PATH_SEEK_LOOP(
      st(a),  // 実行後の状態
      a,      // 実行するアクション
      t,    // 探索開始時のプロセスタイマ
      p,      // 探索開始時からの経過時間
      path+a, // 更新した実行列
      value   // 探索中の価値式
      );
  }
}
```

□

このアルゴリズムは，関数PATH_SEEK_LOOPの再帰呼出により，システムの時間システム上を深さ優先探索することによって，規定の経過時間内に実行可能な全ての実行列を列挙している．なお，制御不能アクションが価値を下げる方向に働くことを考え，実行時間中の最も遅い時間を経過時間として加算していく事で，各実行列の最低価値を導出している．

ステップ2：制約式の導出　求まった実行列と価値式から，制御可能アクションを実行するための制約条件を求める．これは，以下の手順で実行される．

2つ以上の制御可能アクションが実行可能な状態において

1. アルゴリズムによって，その状態から起こり得る全ての実行列を求め，求まった列毎に価値式を求める．
2. 求まった全ての列に対して，以下を実行する．
   (a) 対応する価値式が最大の値を出すための，プロセスタイマに関する条件を導出する．
   (b) (a) で求まった条件を，列の最初に来る制御可能アクションに対応する制約式として追加する．
3. 最終的に出来上がった制約式が，その状態における，リソースの衝突回避を実現する制約式となる．

4 結論

本研究では，Controller systhesis paradigmに基づいたスケジューラ構成法に，"価値関数"とそれに関する要求である"価値要求"を追加し，更に，リソース競合を，システムの価値を最大にする用に解決する衝突回避制約の導出法を考案した．これによって既存のController systhesis paradigmによるスケジューラ構成法では扱えなかった"システムの価値"を記述することで，ソフトリアルタイムシステムのスケジューラを構成できるようにし，それに伴い，"各プロセスに一定以上の価値を保証する"という要求を与えることができるようになり，また，システムの価値を考慮した衝突回避制約を導出することで，よりソフトリアルタイムシステムに適したスケジューラを構成できるようになった．

参考文献

[1] A.M. Tilborg, B.M. Koob. Foundations of Real-time Computing:Formal Specifications and Methods. *Kluwer Academic Pub.*, P.316, 1991.

[2] Jane W. S. Liu. Real-Time Systems. *Prentics-Hall*, 2000.

[3] P.Ramadge, W.Wonham. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3), 637-659, 1987.

[4] K.Altisen, G.Gossler, J.Sifakis Scheduler Modeling Based on the Controller Synthesis Paradigm. *Journal of Real-Time Systems, special issue on "Control Approaches to Real-Time Computing"* , 23, 55-84, 2002.

[5] D.Jensen, C.D.Locke, H.Tokuda. A Time-Driven Scheduling Modes for Real-Time Operating Systems. *IEEE Real-Time Systems Symposium*, pp.112-122, 1985.

[6] H.Kwak, I.Lee, A.Philippou, J.Choi and O.Sokolsky. Symbolic schedulability analysis of real-time systems. *Proceedings*, 409-418, 1999.

[7] H.Wong-Toi, G.Hoffmann. The Control of Dense Real-Time Discrete Event Systems. *Technical report STAN-CS-92-1411*, Stanford University, 1992.

[8] O.Maler, A.Pnueli, J.Sifalis. On the Synthesis of Discrete Controllers for Timed Systems. *STACS'95, LNCS 900*, 229-242, Springer, 1995.

[9] R.Alur, D.Dill. A theory of timed automata. *Theoretical Computer Science*, Vol.126, No.2, 183-236, 1994.

[10] R.Alur, C Courcoubetis, N.Halbwachs, T.Henzinger, P.Ho, X.Nicollin, A.Olibero, J.Sifakis, S.Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138, 3-34, 1995.

[11] Thomas Hune, Kim G. Larsen, Paul Pettersson. Guided Synthesis of Control Programs Using Uppaal. *ICDCS Workshop on Distributed System Validation and Verification*, 15-22, 2000.

[12] R. Alur, C. Courcoubetis, D.L. Dill. Model-checking in dense real-time. *Information and Computation 104(1)*, 2-34, 1993.

[13] R. Alur, C. Courcoubetis, T.A. Henzinger. Computing accumulated delays in real-time systems. *LNCS 697*, pp.181-193, 1993.

[14] Oded Maler, Amir Pnueli, Joseph Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. *LNCS 900*, pp.229-242, 1995.